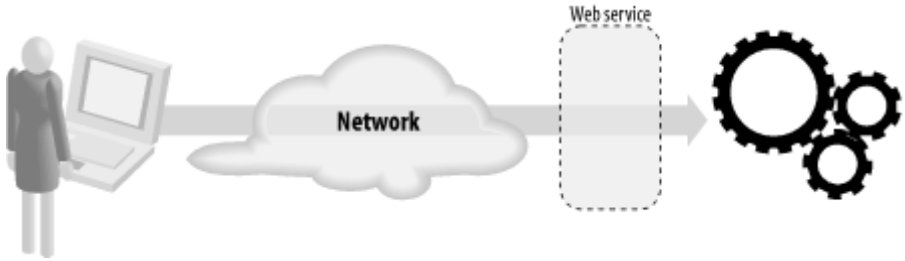
Chapter 1. Introducing Web Services

To make best use of web services and SOAP, you must have a firm understanding of the principles and technologies upon which they stand. This chapter is an introduction to a variety of new technologies, approaches, and ideas for writing web-based applications to take advantage of the web services architecture. SOAP is one part of the bigger picture described in this chapter, and you'll learn how it relates to the other technologies described in this book: the Web Service Description Language (WSDL), the Web Service Inspection Language (WS-IL), and the Universal Description, Discovery, and Integration (UDDI) services.

1.1 What Is a Web Service?

Before we go any further, let's define the basic concept of a "web service." A *web service* is a network accessible interface to application functionality, built using standard Internet technologies. This is illustrated in Figure 1-1.

Figure 1-1. A web service allows access to application code using standard Internet technologies



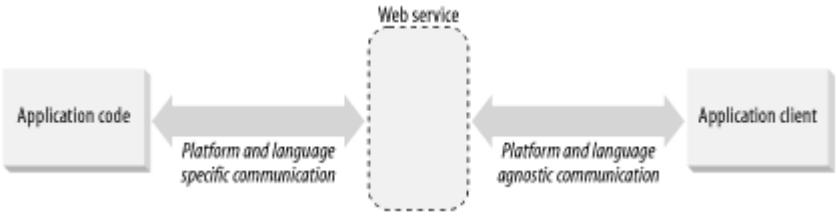
In other words, if an application can be accessed over a network using a combination of protocols like HTTP, XML, SMTP, or Jabber, then it is a web service. Despite all the media hype around web services, it really is that simple.

Web services are nothing new. Rather, they represent the evolution of principles that have guided the Internet for years.

1.2 Web Service Fundamentals

As Figure 1-1 and Figure 1-2 illustrate, a web service is an interface positioned between the application code and the user of that code. It acts as an abstraction layer, separating the platform and programming-language-specific details of how the application code is actually invoked. This standardized layer means that any language that supports the web service can access the application's functionality.

Figure 1-2. Web services provide an abstraction layer between the application client and the application code



The web services that we see deployed on the Internet today are HTML web sites. In these, the application services—the mechanisms for publishing, managing, searching, and retrieving content—are accessed through the use of standard protocols and data formats: HTTP and HTML. Client applications (web browsers) that understand these standards can interact with the application services to perform tasks like ordering books, sending greeting cards, or reading news.

Because of the abstraction provided by the standards-based interfaces, it does not matter whether the application services are written in Java and the browser written in C++, or the application services deployed on a Unix box while the browser is deployed on Windows. Web services allow for cross-platform interoperability in a way that makes the platform irrelevant.

Interoperability is one of the key benefits gained from implementing web services. Java and Microsoft Windows-based solutions have typically been difficult to integrate, but a web services layer between application and client can greatly remove friction.

There is currently an ongoing effort within the Java community to define an exact architecture for implementing web services within the framework of the Java 2 Enterprise Edition specification. Each of the major Java technology providers (Sun, IBM, BEA, etc.) are all working to enable their platforms for web services support.

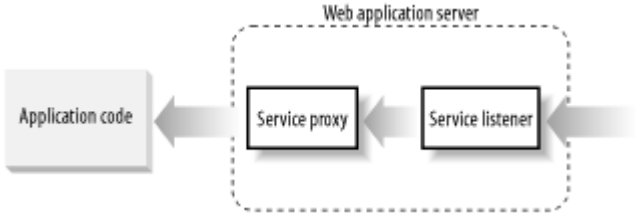
Many significant application vendors such as IBM and Microsoft have completely embraced web services. IBM for example, is integrating web services support throughout their WebSphere, Tivoli, Lotus, and DB2 products. And Microsoft's new .NET development platform is built around web services.

1.2.1 What Web Services Look Like

Web services are a messaging framework. The only requirement placed on a web service is that it must be capable of sending and receiving messages using some combination of standard Internet protocols. The most common form of web services is to call procedures running on a server, in which case the messages encode "Call this subroutine with these arguments," and "Here are the results of the subroutine call."

Figure 1-3 shows the pieces of a web service. The application code holds all the business logic and code for actually doing things (listing books, adding a book to a shopping cart, paying for books, etc.). The Service Listener speaks the transport protocol (HTTP, SOAP, Jabber, etc.) and receives incoming requests. The Service Proxy decodes those requests into calls into the application code. The Service Proxy may then encode a response for the Service Listener to reply with, but it is possible to omit this step.

Figure 1-3. A web service consists of several key components



The Service Proxy and Service Listener components may either be standalone applications (a TCP-server or HTTP-server daemon, for instance) or may run within the context of some other type of application server. As an example, IBM's WebSphere Application Server includes built-in support for receiving a SOAP message over HTTP and using that to invoke Java applications deployed within WebSphere. In comparison, the popular open source Apache web server has a module that implements SOAP. In fact, there are implementations of SOAP for both the Palm and PocketPL Portable Digital Assistant (PDA) operating systems.

Keep in mind, however, that web services do not require a server environment to run. Web services may be deployed anywhere that the standard Internet technologies can be used. This means that web services may be hosted or used by anything from an Application Service Provider's vast server farm to a PDA.

Web services do not require that applications conform to a traditional client-server (where the server holds the data and does the processing) or n-tier development model (where data storage is separated from business logic that is separated from the user interface), although they are certainly being heavily deployed within those environments. Web services may take any form, may be used anywhere, and may serve any purpose. For instance, there are strong crossovers between peer-to-peer systems (with decentralized data or processing) and web services where peers use standard Internet protocols to provide services to one another.

1.2.2 Intersection of Business and Programming

Because a web service exposes an application's functionality to any client in any programming language, they raise interesting questions in both the programming and the business world.

Programmers tend to raise questions like, "How do we do two-phase commit transactions?" or "How do I do object inheritance?" or "How do I make this damn thing run faster?"—questions typically associated with going through the steps of writing code.

Business folks, on the other hand, tend to ask questions like, "How do I ensure that the person using the service is really who they say they are?" or "How can we tie together multiple web services into a workflow?" or "How can I ensure the reliability of web service transactions?" Their questions typically address business concerns.

These two perspectives go hand-in-hand with one another. Every business issue will have a software-based solution. But the two perspectives are also at odds with each other: the business processes demand completeness, trust, security, and reliability, which may be incompatible with the programmers' goals of simplicity, performance, and robustness.

The outcome is that tools for implementing web services will do so from one of these two angles, but rarely will they do so from both. For example, SOAP::Lite, the Perl-based SOAP implementation written by the coauthor of this book, Pavel Kulchenko, is essentially written for programmers. It provides a very simple set of tools for invoking Perl modules using SOAP, XML-RPC, Jabber, or any number of other protocols.

In contrast, Apache's Axis project (the next generation of Apache's SOAP implementation) is a more complex web services implementation designed to make it easier to implement processes, or to tie together multiple web services. Axis can perform the stripped down bare essentials, but that is not its primary focus.

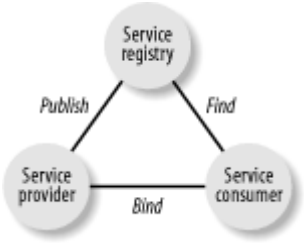
The important thing to keep in mind is that both tools implement many of the same set of technologies (SOAP, WSDL, UDDI, and others, many of which we discuss later on), and so they are capable of interoperating with each other. The differences are in the way they interface with applications. This gives programmers a choice of how their web service is implemented, without restricting the users of that service.

1.2.3 Just-In-Time Integration

Once you understand the basic web services outlined earlier, the next step is to add *Just-In-Time Integration*. That is, the dynamic integration of application services based not on the technology platform the services are implemented in, but upon the business requirements of what needs to get done.

Just-In-Time Integration recasts the Internet application development model around a new framework called the web services architecture (Figure 1-4).

Figure 1-4. The web services architecture



In the web services architecture, the *service provider* publishes a description of the service(s) it offers via the *service registry*. The *service consumer* searches the service registry to find a service that meets their needs. The service consumer could be a person or a program.

Binding refers to a service consumer actually using the service offered by a service provider. The key to Just-in-Time integration is that this can happen at any time, particularly at runtime. That is, a client might not know which procedures it will be calling until it is running, searches the registry, and identifies a suitable candidate. This is analogous to late binding in object-oriented programming.

Imagine a purchasing web service, where consumers requisition products from a service provider. If the client program has hard-coded the server it talks to, then the service is bound at compile-time. If the client program searches for a suitable server and binds to that, then the

service is bound at runtime. The latter is an example of Just-In-Time integration between services.

1.3 The Web Service Technology Stack

The web services architecture is implemented through the layering of five types of technologies, organized into layers that build upon one another (Figure 1-5).

Figure 1-5. The web service technology stack

It should come as no surprise that this stack is very similar to the TCP/IP network model used to describe the architecture of Internet-based applications (Figure 1-6).

Figure 1-6. The TCP/IP network model

The additional packaging, description, and discovery layers in the web services stack are the layers essential to providing Just-In-Time Integration capability and the necessary platform-neutral programming model.

Because each part of the web services stack addresses a separate business problem, you only have to implement those pieces that make the most sense at any given time. When a new layer of the stack is needed, you do not have to rewrite significant chunks of your infrastructure just to support a new form of exchanging information or a new way of authenticating users.

The goal is total modularization of the distributed computing environment as opposed to recreating the large monolithic solutions of more traditional distributed platforms like Java, CORBA, and COM. Modularity is particularly necessary in web services because of the rapidly evolving nature of the standards. This is shown in the sample CodeShare application of Chapter 7, where we don't use the discovery layer, but we do draw in another XML standard to handle security.

1.3.1 Beyond the Stack

The layers of the web services stack do not provide a complete solution to many business problems. For instance, they don't address security, trust, workflow, identity, or many other business concerns. Here are some of the most important standardization initiatives currently being pursued in these areas:

XML Protocol

The W3C XML Protocol working group is chartered with standardizing the SOAP protocol. Its work will eventually replace the SOAP protocol completely as the de facto standard for implementing web services.

XKMS

The XML Key Management Services are a set of security and trust related services that add Private Key Infrastructure (PKI) capabilities to web services.

SAML

The Security Assertions Markup Language is an XML grammar for expressing the occurrence of security events, such as an authentication event. Used within the web services architecture, it provides a standard flexible authentication system.

XML-Dsig

XML Digital Signatures allow any XML document to be digitally signed.

XML-Enc

The XML Encryption specification allows XML data to be encrypted and for the expression of encrypted data as XML.

XSD

XML Schemas are an application of XML used to express the structure of XML documents.

P3P

The W3C's Platform for Privacy Preferences is an XML grammar for the expression of data privacy policies.

WSFL

The Web Services Flow Language is an extension to WSDL that allows for the expression of work flows within the web services architecture.

Jabber

Jabber is a new lightweight, asynchronous transport protocol used in peer-to-peer applications.

ebXML

ebXML is a suite of XML-based specifications for conducting electronic business. Built to use SOAP, ebXML offers one approach to implementing business-to-business integration services.

1.3.2 Discovery

The discovery layer provides the mechanism for consumers to fetch the descriptions of providers. One of the more widely recognized discovery mechanisms available is the Universal Description, Discovery, and Integration (UDDI) project. IBM and Microsoft have jointly proposed an alternative to UDDI, the Web Services Inspection Language (WS-Inspection). We will discuss both UDDI and WS-Inspection in depth (including arguments for and against their use) in Chapter 6.

1.3.3 Description

When a web service is implemented, it must make decisions on every level as to which network, transport, and packaging protocols it will support. A description of that service represents those decisions in such a way that the Service Consumer can contact and use the service.

The Web Service Description Language (WSDL) is the de facto standard for providing those descriptions. Other, less popular, approaches include the use of the W3C's Resource Description Framework (RDF) and the DARPA Agent Markup Language (DAML), both of which provide a much richer (but far more complex) capability of describing web services than WSDL.

We cover WSDL in Chapter 5. You can find out more information about DAML and RDF from:

http://daml.semanticweb.org/
http://www.w3.org/rdf

1.3.4 Packaging

For application data to be moved around the network by the transport layer, it must be "packaged" in a format that all parties can understand (other terms for this process are "serialization" and "marshalling"). This encompasses the choice of data types understood, the encoding of values, and so on.

HTML is a kind of packaging format, but it can be inconvenient to work with because HTML is strongly tied to the presentation of the information rather than its meaning. XML is the basis for most of the present web services packaging formats because it can be used to represent the meaning of the data being transferred, and because XML parsers are now ubiquitous.

SOAP is a very common packaging format, built on XML. In Chapter 2, we'll see how SOAP encodes messages and data values, and in Chapter 3 we'll see how to write actual web services with SOAP. There are several XML-based packaging protocols available for

developers to use (XML-RPC for instance), but as you might have guessed from the title of this book, SOAP is the only format we cover.

1.3.5 Transport

The transport layer includes the various technologies that enable direct application-to-application communication on top of the network layer. Such technologies include protocols like TCP, HTTP, SMTP, and Jabber. The transport layer's primary role is to move data between two or more locations on the network. Web services may be built on top of almost any transport protocol.

The choice of transport protocol is based largely on the communication needs of the web service being implemented. HTTP, for example, provides the most ubiquitous firewall support but does not provide support for asynchronous communication. Jabber, on the other hand, while not a standard, does provide good a asynchronous communication channel.

1.3.6 Network

The network layer in the web services technology stack is exactly the same as the network layer in the TCP/IP Network Model. It provides the critical basic communication, addressing, and routing capabilities.

1.4 Application

The application layer is the code that implements the functionality of the web service, which is found and accessed through the lower layers in the stack.

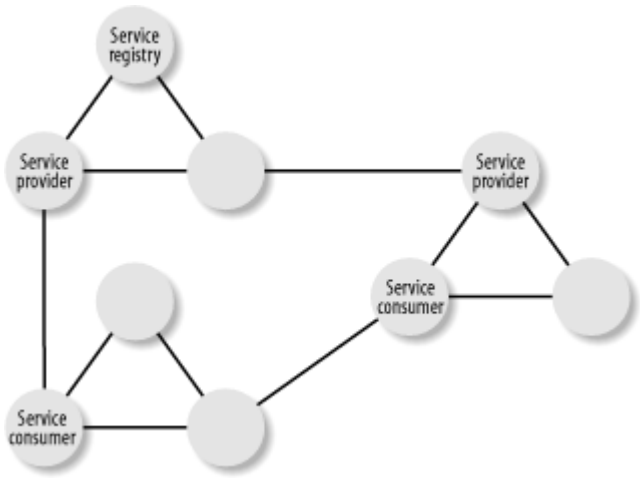
1.5 The Peer Services Model

The peer services model is a complimentary but alternative view of the web services architecture. Based on the peer-to-peer (P2P) architecture, every member of a group of peers shares a common collection of services and resources. A peer can be a person, an application, a device, or another group of peers operating as a single entity.

While it may not be readily apparent, the same fundamental web services components are present as in the peer services architecture. There are both service providers and service consumers, and there are service registries. The distinction between providers and consumers, however, is not as clear-cut as in the web services case. Depending on the type of service or resource that the peers are sharing, any individual peer can play the role of both a service provider and a service consumer. This makes the peer services model more dynamic and flexible.

Instant Messaging is the most widely utilized implementation of the peer services model. Every person that uses instant messaging is a peer. When you receive an invitation to chat with somebody, you are playing the role of a service provider. When you send an invitation out to chat with somebody else, you are playing the role of a service consumer. When you log on to the Instant Messaging Server, the server is playing the role of the service registry—that is, the Instant Messaging Server keeps track of where you currently are and what your instant messaging capabilities are. Figure 1-7 illustrates this.

Figure 1-7. The peer web services model simply applies the concepts of the web services architecture in a peer-to-peer network



Peer services and web services emerged and evolved separately from one another, and accordingly make use of different protocols and technologies to implement their respective models. Peer web services tie the two together by unifying the technologies, the protocols, and the models into a single comprehensive big picture. The implementation of a peer web service will be the central focus of Chapter 7.